Paralelisasi Klasifikasi Data Ekspresi Gen Kanker dengan Algoritma Deep Neural Network Menggunakan Stacked Sparse Autoencoder

Aswindo Putra[1], Jondri[2], Fitriyani[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]aswindoputra@student.telkomuniversity.ac.id, [2]jondri@telkomuniversity.ac.id,
[3]fitriyani@telkomuniversity.ac.id

Penelitian bidang bioinformatika menjadi populer saat ini sebagai solusi bagi dunia medis. Salah satunya klasifikasi penyakit kanker menggunakan data g*ene expression*. *Deep learning* telah menjadi penelitian yang menarik pada bidang bioinformatika. Banyak penelitian tentang klasifikasi kanker yang diangkat menggunakan *deep learning*. Klasifikasi menggunakan data *gene expression* berguna di dunia medis. Karena dapat mengklasifikasikan penyakit hanya menggunakan gen. Penelitian ini mengangkat klasifikasi gene kanker menggunakan metode *deep neural network* dengan *stacked spare autoencoder* dan *autoencoder* sebagai metode extraksinya. Selain itu digunakan juga sparse autoencoder sebagai representasi dari pembelajaran neural network. Ini digunakan untuk mengurangi masalah saat pembelajaran. *Fine-tune* digunakan sebagai optimasi bobot dan bias untuk jaringan *neural network* dengan metode *gradient descent*. Pengklasifikasian hasil dari pembelajaran menggunakan *softmax classifier*. Data yang digunakan bersumber dari *portal of National Center for Biotechnology Information*. Jumlah dataset yang digunakan sebanyak 1065 sampel dari 8 kategori kelas untuk beberapa penyakit kanker dan non kanker. Dengan ini diperoleh hasil akurasi tertinggi 97,3 % untuk *training* dan 92,6 % untuk *testing*. Paralelisasi dari algoritma ini dapat bekerja dengan baik, dimana efesiensi terhadap waktu komputasi lebih cepat dengan *speed up* sekitar 13,03 terhadap komputasi sekuensial. Tentunya ini menjadi momentum untuk mengembangkan algoritma *neural network* lainnya dengan teknik paralelisasi.

Kata Kunci : Deep Learning, SSAE, Parallel Computing, High Performance Computing, Cancer Classification, gene expression

Nowadays, research in bioinformatics can be one of the popular researches for the medical professional researcher. One of them is the classification using data gene expression. Deep learning has become an interesting research in bioinformatics fields. Many kinds of research about cancer classification are appointed using deep learning. Classification using gene expression dataset is useful in the medical profession. Because it can classify diseases using gene only. This research raised the classification gene expression cancer using deep neural network with stacked sparse autoencoder and autoencoder as an extraction method. In addition, sparse autoencoder is also used as a represented of neural network learning. Fine tuning is used as weight and bias optimization for neural network with gradient descent method. Classification of the learning result using the softmax classifier. This research, the data is sourced from the portal of National Center for Biotechnology Information. With this obtained, the highest accuracy of 97.3% for training and 92,6 % for testing. Parallelization for this algorithm work quite well, this shows to the efficiency of the computation time is faster with speed up 000% for sequential computation. Of course, this research becomes the best moment for other neural network algorithm development with parallelism technique.

Keyword: Deep Learning, SSAE, Parallel Computing, High Performance Computing, Cancer Classification, gene expression

I. Pendahuluan

Berkembangnya metode *deep learning* pada dunia kesehatan meningkatkan nilai metode ini sebagai topik dalam penelitian terbaik untuk analisis problem di dunia kesehatan. Sebagai perbandingan penggunaan *deep learning* terbesar saat ini berasal dari bidang bioinformatika [6]. Banyak metode *deep learning* digunakan untuk penelitian yang berhubungan dengan bioinformatika dan kesehatan. Ini menandakan metode *deep learning* menjadi metode analisis yang terus dikembangankan pada bidang bioinformatika.

Sebagai solusi yang mendekati untuk proses analisis terbaik, *deep learning* menawarkan metode terbaik dalam menganalisis masalah. Pendekatan pembelajaran mendalam menjadi nilai tambah bagi kebanyakan peneliti di bidang bioinformatika untuk mengembangkan *deep learning* sejauh mungkin. *Deep learning* menawarkan banyak metode untuk memberikan solusi dalam menyelesaikan masalah. Salah satunya metode *deep nueural network* **(DNN)** yang memodelkan fitur tingkat tinggi dan diabstraksikan menggunakan beberapa lapisan non linear yang kompleks [2].

Sebagai metode yang dapat mempelajari pola pola unik, *deep learning* dapat belajar secara mendalam terhadap sebuah pola yang dipelajari. Pada kasus ini *gene* memiliki rantai pola tertentu yang dapat dipelajari oleh

sistem pembelajaran mendalam. *Gene* memiliki pola unik yang tersusun dalam sebuah array, hanya dapat dipelajari dengan metode khusus. Untuk memperoleh pendekatan terbaik, DNN dapat menjadi solusi untuk menentukan sebuah solusi dari klasifikasi penyakit kanker.

Besarnya volume data yang dipelajari, membuat pembelajaran menjadi semakin akurat karena banyak pola yang dapat dipelajari, tetapi akan memperlama waktu komputasi karena dibutuhkan banyak sumber daya untuk mempelajari data dengan volume besar. Sebagai solusi untuk menghemat waktu komputasi, pembelajaran secara paralel ditawarkan untuk pelatihan dengan data besar sehingga mempersingkat waktu komputasi.

Bagian dari penelitian yaitu menyelesaikan masalah yang harus dipecahkan untuk menemukan solusi dari hasil penelitian. Adapun perumusan masalah yang diangkat dari penelitian ini terbagi atas tiga yaitu, (a). mengklasifikasikan penyakit kanker dari data gen dengan metode DNN menggunakan *stacked sparse autoencoder* (SSAE) , (b).implementasi dari algoritma *training* SSAE pada metode DNN untuk memperoleh akurasi *training* yang tinggi, dan (c). paralelisasi DNN sehingga diperoleh efisiensi waktu komputasi program serta melakukan komparasi dengan membandingkan waktu komputasi program sekuensial dan paralel.

Dengan identifikasi masalah pada penelitian ini dapat diambil batasan untuk cakupannya yaitu klasifikasi penyakit kanker menggunakan metode DNN dengan metode ekstraksinya menggunakan *autoencoder*. Selain itu, implementasi algoritma *training* dari SSAE pada arsitektur DNN untuk mendapatkan akurasi *training* yang tinggi menjadi batasan pada penelitian ini. Serta cara paralelilasi algoritma tersebut untuk mendapatkan efesiensi waktu komputasi tercepat dengan komputasi paralel.

Merujuk pada identifikasi dan batasan masalah dalam penelitian ini, maka dapat diuraikan tujuan untuk penelitian. Penelitian ini memiliki 3 tujuan yaitu (a). untuk mendapatkan hasil pembelajaran terbaik dari klasifikasi penyakit kanker menggunakan DNN, (b). mengamati pengaruh *hyperparameter* DNN terhadap akurasi pembelajaran, dan (c.) mempresentasikan waktu komputasi dari komparasi algoritma sekuensial dan paralel untuk algoritma DNN bahwa komputasi paralel memberikan efesiensi yang besar terhadap beban komputasi yang lama.

Penulisan pada jurnal ilmiah ini dibagi menjadi 5 bagian diantaranya bagian I Pendahuluan, II Studi terkait, III Perancangan Sistem, IV Evaluasi, dan bagian terakhir V.Kesimpulan.

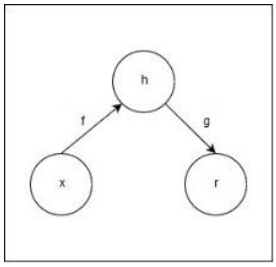
II. Studi Terkait

Jaringan syaraf tiruan (**JST**) atau *neural network* (**NN**) merupakan sebuah jaringan dengan konsep sederhana yang dimodelkan berdasarkan jaringan syaraf manusia untuk menemukan pola unik dari data yang dilatih. JST merupakan sistem dasar dari sistem *deep learning*. Sistem adaptif dari JST dapat mengubah dan memecahkan masalah berdasarkan informasi yang mengalir pada jaringan tersebut [5]. Pada ruang lingkup JST, dikenal dengan sebutan *neuron* merupakan sebuah sistem syaraf pada otak manusia. JST dapat didefenisikan sebagai $f: X \rightarrow Y$ merupakan istilah untuk mengambarkan interkoneksi dari beberapa jaringan *neuron* yang berada pada posisinya masing masing [5].

Autoencoder

Autoencoder adalah konsep lanjutan dari JST yang digunakan untuk pembelajaran tanpa pengawasan atau disebut *unsurpervised learning*. Pembelajaran *autoencoder* untuk mempelajari representasi data yang digunakan sebagai pengurangan dimensi dataset [8]. Sebelum proses pembelajaran, *autoencoder* digunakan sebagai pereduksi dimensi dataset dengan fitur ekstraksi [9]. *Autoencoder* merupakan representasi dari JST yang dilatih untuk menyalin input ke output.

Dalam arsitekturnya, *autoencoder* memiliki lapisan tersembunyi disebut dengan, h, yang menggambarkan kode untuk mewakili masukannya. Arsitektur dari *autoencoder* terdiri dari dua bagian yaitu (a). sebagai fungsi *encoder* bertugas untuk merubah nilai input x menjadi vektor, didefenisikan dengan model $h - f(x)$ dimana $f(x)$ sebagai fungsi *encoder* dan, (b). sebagai *decoder* untuk membentuk kembali vektor menjadi input dan menghasilkan vektor dengan model kontruski $r - g(h)$ dimana $g(h)$ sebagai fungsi decoder.



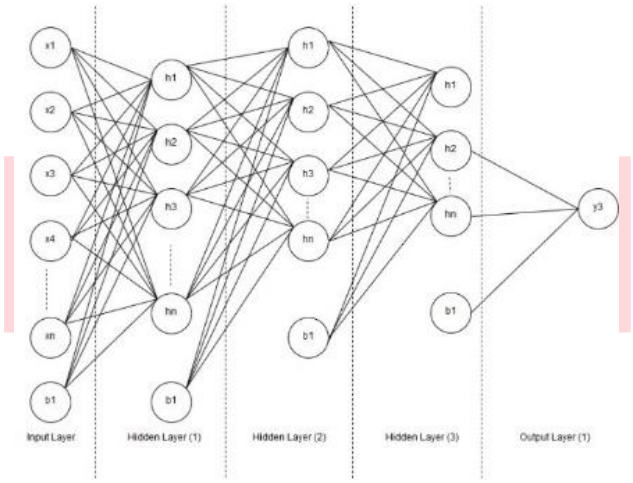
Gambar 1. Struktur umum arsitektur *autoencoder* yang memetakan input x ke ouput r melalui *hidden layer* dengan representasi internal h. [13]

Penerapan algoritma *backpropagation* dapat terjadi pada metode *autoencoder*. Pada dasarnya *autoencoder* tidak dapat melakukan pembelajaran secara sempurna. Model ini sendiri lebih memperhatikan apsek yang akan

menjadi inputan untuk pembelajaran. Karena itu model ini tidak dapat menyalin secara sempurna setiap masukan disebabkan fungsi utama dari autoencoder sebagai reduksi dimensi dataset [9][13].

Stacked Sparse Autoencoder

SSAE merupakan sebuah JST yang terdiri dari sejumlah lapisan tersembunyi dari sebuah *sparse autoencoder* (**SAE**), output dari lapisan tersembunyi terhubung ke lapisan input secara berturut turut [10]. Pendekatan *greedy layerwise* dapat dijadikan sebagai *pre-trainning* dalam pelatihan setiap lapisan secara bergantian. *Autoencoder* dapat ditumpuk berlapis untuk melakukan *pre-training* dengan menginisialisasikan setiap bobot pada jaringan. *Autoencoder* yang ditumpuk merupakan JST yang terdiri dari beberapa lapisan alat ukur otomatis tipis, dimana keluaran setiap lapisan dihubungkan pada input secara berturut turut.



Gambar 2. Fitur tingkat tinggi menggunakan SSAE.

Arsitektur *autoencoder* membuat dengan pendekatan tiga lapisan SAE yang menjadi pertimbangan, yang terdiri dari tiga lapisan tersembunyi. SSAE memiliki lapisan SAE untuk fitur pembelajaran tingkat tinggi sesuai dengan Gambar 2 Jaringan syaraf *feedforward* melakukan *training* secara efektif oleh satu lapisan pada satu waktu. Pembelajaran *unsurpervised learning* dengan teknik *greedy* dapat meningkatkan kinerja dan inisialisasi bobot [10].

Sparse Autoencoder

Bagian yang sederhana dari SAE merupakan lapisan input dan output dengan lapisan tersembunyi untuk menghubungkan kedua lapisan tersebut. Pendekatan ini tidak sendirinya bersaing dengan fitur *hand-engginered* terbaik, tetapi fitur yang bisa dipelajari berguna untuk banyak masalah. *Autencoder*, mempelajari fungsi hipotesis $h_{w,b}(x) = x$ dengan data tidak berlabel dimana b merupakan matrix bobot dan vektor bias topologi jaringan masing masing. [10]. Pelatihan tidak diberi label maka fungsi biaya menjadi tiga bagian yaitu *hypothesis error*, *weight decay*, dan *sparsity penalty*. Model dalam bentuk matematisnya akan dijelaskan di bagian *training deep neural network* sehingga didapatkan persamaan dari $J_{sparse}(W, b)$.

Training Deep Neural Network with Stacked Sparse autoencoder

Berdasarkan ilustrasi dari algoritma 1 pembelajaran SAE tunggal dimana bobot dari DNN diinisialisasi dengan melatih suku dari *autencoder* dengan *layerwise* [10]. Pada kasus ini terlihat bobot antara *input* dan *hidden layer* awal diinisialisasi dengan proses pembelajaran SAE yang memiliki kerangka kerja sesuai dengan *input* dan *hidden layer* awal [10]. Untuk menghasilkan representasi fitur tingkat pertama dari dataset maka proses pelatihan dari data pelatihan input akan digantikan kedepan hingga lapisan tersembunyi pertama. Setelah diperoleh data dari fitur tingkat pertama maka langkah selanjutnya adalah analisa bobot lapisan tersembunyi pertama dan kedua dengan mempelajari SAE dari kerangka kerja yang sama.

Algoritma 1. Proses belajar dari *spare autoecoder*

1. Inisialiasi bobot dan bias diantara $[-\tau, \tau]$, dimana τ konstan terhadap jumlah *nueron* pada lapisan tersembunyi.
2. Inisialisasi $\Delta \boldsymbol{W}^l = 0$, dan $\Delta \boldsymbol{b}^l = 0$ untuk semua layer.
3. Komputasi encoder pada hidden layer.

$$z = W.data + b$$
$$a = f(x)$$

4. Weight transpose.

$$W = W.T$$

5 Komputasi decoder pada hidden layer

$$z = W.data + b$$
$$a = f(x)$$

6 Rekontruksi Input

7 Menerapkan algoritma *backpropagation*

$$\Delta W^l := \Delta W^l + \nabla_{W^l} J(W, b; x, y)$$
$$\Delta b^l := \Delta b^l + \nabla_{b^l} J(W, b; x, y)$$

8 Melakukan update matrix bobot W dan vektor bias b.

$$\boldsymbol{W^l} = W^l - \alpha \left[\frac{1}{m}\Delta(W^l) + \lambda(W^l)\right]$$
$$\boldsymbol{b^l} = b^l - \alpha \left[\frac{1}{m}\Delta(b^l)\right]$$

Dimana α dan m merupakan learning rate dan jumlah dataset

Proses inisialisasi dilakukan sampai lapisan terakhir dengan menumpuk SAE setiap lapisan. DNN akan disetel berdasarkan tujuannya sesuai dengan Gambar 2.2 yang telah diilustarsikan. Setelah itu langkah selanjutnya menentukan fungsi biaya dan dari klasifikasi. Jika tahapan ini telah selesai maka langkah selanjutnya diteruskan dengan proses pembelajaran *bakcpropagation* secara lengkap.

Forward Pass

Pada tahap awal inisialisasi parameter bobot dan bias secara acak, kemudian akan dibawa pada nilai awal untuk melatih *autoencoder* yang ditumpuk. Setiap inputan yang masuk ke dalam arsitektur dari DNN maka dikalkulasikan dengan bobot di setiap *layer* dan akan menjadi inputan untuk lapisan selanjutnya. Jika menggunakan autoencoder, terdapat dua tahap yang dilewati oleh data yaitu tahap *encoding* dan tahap *decoding*. Secara matematis tahap encoding dapat digambarkan dalam persamaan (1) :

$$a^{(i)} = f\left(z^{(i)}\right);\ z^{(i)} = \sum_{i=1}^{n}\sum_{j=1}^{m}(W^{(i,j)}.X^{(j)}) + B^{(i)}$$

Persamaan 1

dimana z merupakan penjumlahan dari seluruh neuron pada layer ke i yang akan menjadi masukan untuk fungsi aktivasi. Fungsi aktivasi dalam NN memiliki banyak variasi, tergantung pada nilai input yang digunakan. Jika nilai *input features* pada *range* 0 sampai 1 maka fungsi aktivasi yang cocok adalah sigmoid. Sigmoid digunakan untuk mengaktifkan sel neuron pada setiap layer dalam arsitektur NN yang memiliki nilai pada *range* 0 sampai 1. Fungsi aktivasi sigmoid dapat didefenisikan pada persamaan (2) [17].

$$y = a = f\ (x) = \frac{1}{(1 + e^{-x}\ )}$$

Persamaan 2

Setelah tahap encoding selesai, maka bobot di transpose dan dikalkulasikan dengan neuron *decoder* untuk mendapatkan rekontruksi input, yang didefenisikan pada persamaan (3) [17][10]:

$$a^{(i)} = f\left(z^{(i)}\right);\ z^{(i)} = \sum_{i=1}^{n}\sum_{j=1}^{m}(W^{T(i,j)}.X^{(j)}) + B^{(i)}$$

Persamaan 3

Pembetukan nilai input kembali dari proses rekontruksi input memungkinkan arsitektur untuk menemukan nilai dari error pada setiap pelatihan. Error digunakan untuk melihat sebarapa jauh kesalahan yang terjadi saat *training*. Perhitungan error sejalan dengan *cost function* dan didefenisikan pada persamaan (4) :

$$J(W, b; x, y) = \frac{1}{2}\left\|h_{W,b}(x^i) - y^i\right\|^2$$

Persamaan 4

dimana $h_{W,b}$ merupakan hypothesis dari output dalam unit partikular. *hypothesis* dari output didefenisikan pada persamaan (5) :

$$h_{W,b} = f(\sum_{i=1}^{n} W_i X_i + b\ ) = a$$

Persamaan 5

Disisi lainya $\left\|h_{W,b}(x^i) - y^i\right\|$ merupakan jarak *euclidean* antara *hypothesis* output dan ouput. Untuk mendefenisikan kesalahan dari *sum of squares* terhadap semua sampel pelatihan, maka fungsi biaya *sparse* dapat didefenisikan pada persamaan (6) [17]

$$J_{sparse}(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left\|h_{W,b}(x^i) - y^i\right\|^2\right]$$

Persamaan 6

Penghitungan kerusakan pada bobot biasanya disebut dengan *weight decay term.* Pada tahap ini terjadi peluruhan bobot yang biasa disebut dengan *regularization term*. Ini digunakan untuk menambahkan biaya kerusakan pada bobot yang akan menjadi penalti untuk *cost function*. Sebagai mana yang dijelaskan [17] bahwa penalti ini berguna untuk mengurangi perdebaan yang berlebihan pada output. Kalkulasi terhadap *weight decay* membutuhkan sebuah kontrol yang disebut dengan *weight decay parameter*, λ. Dengan adanya *weight decay*, maka ini ditambahkan ke *cost function* pada persamaan (6) dan didefenisikan pada persamaan (7) :

$$J_{sparse}(W,b) = J_{sparse}(W,b) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(W_{j,i}^{l}\right)^2$$

Persamaan 7

dimana n_l merupakan jumlah layer dalam arsitektur dan s_l merupakan jumlah neuron dalam sebuah layer.

Sementara itu untuk mewakili aktivitas dari output unit tersembunyi dari *autoencoder* tumpukan yang pertama, maka nilai dari aktivasi secara ekspilit mewakili aktivitas dari unit tersembunyi yang sama dan diberi beberapa masukan tertentu. Sehingga bagian ketiga dari *cost function* yaitu menetapkan batasan *sparsity* dari *autoencoder*. Istilah $\hat{\rho}$ digunakan untuk mewakili aktivasi rata rata dari neuron tersebut pada lapisan tertentu yang akan dihasilkan dari semua masukan ke unit neouron tersebut. Ini didefenisikan pada persamaan (8) :

$$\hat{\rho} = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^2(x^i)\right]$$

Persamaan 8

S*parsity penalty* membutuhkan *sparsity parameter* yang sama dengan $\hat{\rho}$ dan *control sparsity paramter* yang masing masing dilambangkan dengan ρ, β. Untuk mencapai kesetaraan antara $\hat{\rho}$ dan ρ, ditambahkan model Kullback-Leibler untuk penalty $\hat{\rho}$ ketika mengalami penyimpangan signifikan dari ρ. Kullback-leibler mengukur perbedaan antara keduanya dengan distribusi probabilitas, didefenisikan pada persamaan (9) :

$$\sum_{j=1}^{s_l} KL(\hat{\rho}||\rho) = \sum_{j=1}^{s_l}\left(\rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\left(\frac{1-\rho}{1-\hat{\rho}}\right)\right)$$

Persamaan 9

Kullback-leibler atau KL-divergence mengukur informasi yang hilang saat ρ digunakan untuk aproksimasi $\hat{\rho}$ ketika menjumlahan unit dalam *hidden layer*. Sekarang *cost function* secara keseluruhan didefenisikan pada persamaan (10) :

$$J_{spars}(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left\|h_{W,b}(x^i) - y^i\right\|^2\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(W_{j,i}^{l}\right)^2 + \beta\sum_{j=1}^{s_l} KL(\hat{\rho}||\rho)$$

Persamaan 10

Backpropagation

Backpropagation adalah tahap pembelajaran terhadap bobot yang telah menyelesaikan kalkulasi *forward*. Bobot yang digunakan untuk pembelajaran bisa saja tidak sesuai dengan proses pembacaan pola dari *input features*. Optimasi bobot menggunakan metode *backpropagaiton* sehingga didapatkan bobot terbaru untuk memulai pembelajaran selanjutnya. *fine-tune* digunakan untuk mengingkatkan kinerja *stacked of autoencoder*. *Fine-tune* memperlakukan semua lapisan SSAE sebagai jaringan tunggal dalam satu iterasi dan memperbaiki semua bobot pada SSAE.

Teknik untuk memperbaharui bobot dan bias menggunakan konsep *gradient descent* dengan cara mendiferensiasikan *cost function*. Jika menggunakan *autoencoder* maka diferensiasi ditambahkan dengan model dari *sparsity* dengan model cost function pada persamaan (10). Secara matematis dapat didefenisikan pada persamaan (11) dan (12) :

$$\frac{\delta}{\delta W_{ji}^{l}}J_{sparse}(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\delta}{\delta W_{ji}^{l}}J(W,b;x^i,y^i)\right] + \lambda W_{ji}^{l}$$

Persamaan 11

$$\frac{\delta}{\delta b_{i}^{l}}J_{sparse}(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\delta}{\delta b_{i}^{l}}J(W,b;x^i,y^i)$$

Persamaan 12

dimana $\frac{\delta}{\delta w_{ji}^{l}}$ dan $\frac{\delta}{\delta b_{i}^{l}}$ merupakan diferensiasi dari bobot dan bias menggukana metode chain rule[18].

Setelah diperoleh diferensiasi dari bobot dan bias selanjutnya pembaharuan masing masingnya menggunakan metode *gradient descent*. Bobot dan bias yang telah diperbaharui digunakan kembali untuk pembelajaran selanjutnya. Proses pembaharuan bobot dan bias didefenisikan pada persamaan (13) dan (14):

$$W_{ij}^{l}{}^{+} \coloneqq W_{ji}^{l} - \alpha \frac{\delta}{\delta W_{ji}^{l}} J(W,b)$$

Persamaan 13

$$b_{i}^{l}{}^{+} \coloneqq b_{i}^{l} - \alpha \frac{\delta}{\delta b_{i}^{l}} J(W,b)$$

Persamaan 14

dimana α merupakan *learning rate* untuk tingkatan langkah pada saat pembaharuan bobot dan bias menggunakan metode *gradient descent*

Softmax Classifier

Softmax regression merupakan generalisasi regresi logistik untuk khasus yang menangani banyak kelas. Ini merupakan metode diawasi untuk dataset kelas ganda dengan regresi logistik. fW sama dengan persaman (2) merupakan fungsi sigmoid dengan parameter W merupakan bobot dari setiap lapisan. Inputan x merupakan fitur tingkat tinggi yang telah dipelajari oleh SSAE dan parameter W diset dengan pelatihan $\{h^3(k), y\,(k)\}_{m=1}^{N}$ untuk meminimalkan fungsi biaya.Pada proses ini nantinya layer *softmax* akan melakukan update bobot dan bias untuk meminimalkan fungsi biaya dengan algoritma *backprogation*. Selanjutnya diperoleh nilai optimal untuk dapat dipetakan sesuai pada setiap kelasnya.

Komputasi Paralel

Komputasi paralel merupakan komputasi yang dilakukan bersama dengan memanfaatkan banyak mesin atau komputer independen secara bersamaan. Paralelisasi sering digunakan jika terdapat beban algoritma yang besar. Komputasi data besar membutuhkan biaya yang besar jika diproses dengan paralel karena membutuhkan banyak sumber daya. Dalam komputasi paralel dikenal pemrograman paralel sebagai teknik komputasi algoritma secara paralel dengan memanfaatkan banyak pemrosesan. Dengan ini tujuan akhir dari komputasi adalah meringkas waktu dan meningkatkan performance komputasi.

Performasi komputasi paralel dapat diukur dari pengingkatan kecepatannya yang disebut dengan *speed up*, diperoleh saat membandingakan sebuah program parelel dengan program sekuensial. Peningkatan kecepatan pada program paralel dapat didefenisikan pada persamaan (14):

$$\psi(n,p) = \frac{T_s}{T_p}$$

Persamaan 15

dengan ψ adalah *speed up* dari peningkatan algoritma paralel terhadap algoritma sekuensial, serta T_s dan T_p masing masing nya adalah waktu eksekusi program sekuensial dan paralel serta p jumlah dari prosesor. Selain itu, performansi komputasi paralel juga dihitung berdasarkan efisiensi terhadap terhadap biaya komputasi, dapat didefenisikan pada persamaan(15):

$$\varepsilon(n,p) = \frac{\psi(n,p)}{p} . 100\ \%$$

Persamaan 16

Amdahl Law

Hukum Amdahl merupakan formulasi peningkatan kecepatan yang teoritis dalam latensi pengerjaan tugas besar berdasarkan beban kerja yang tetap yang diharapkan dari sumber daya yang ditingkatkan. Hukum Amdahl sering digunakan untuk menghitung peningkatan kecepatan pada komputasi paralel saat mengerjakan program dengan banyak komputer atau *core*[19]. Untuk mengetahui *speed up* berdasarkan *Amdahl law* dapat didefenisikan pada persamaan (16) :

$$\psi_{amdahl}(n,p) = \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \frac{\varphi(n)}{p}} \; ; 0 < \alpha < 1$$

Persamaan 17

dengan σ dan φ merupakan porsi sekuensial dan paralel.

III. Perancangan Sistem

Dalam penelitian ini, dataset yang diambil merupakan data sekunder yang diperoleh dari halaman website penelitian bioinformatika pada *portal of National Center for Biotechnology Information*. Portal ini merupakan portal dari *United States National Library of Medicine* (**NLM**) di Bethaseda, Maryland, United States of America.

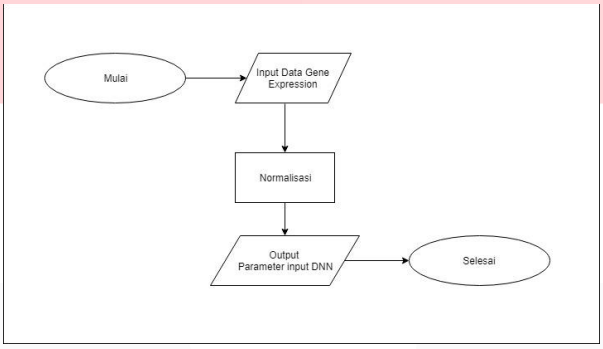
Berkas yang digunakan yaitu hasil pemrosesan dataset dari *repository* yang telah ada. Data diproses sebelum dijadikan data inputan. Sebagai data untuk *training*, maka diambil beberapa data penyakit kanker dan

dibagi menjadi beberapa kelas seperti yang dijelaskan pada tabel 1 dengan jumlah 1065 sampel dari 8 kategori kelas. Selain itu digunakan juga data non kanker untuk klasifikasi.

Tabel 1. Daftar data dan kelas pembeda

No	Nama Data	Kelas	Jumlah (sampel)
1	Breast Cancer	0	137
2	Prostate Cancer	1	235
3	Colon Cancer	2	203
4	Endometrium	3	188
5	Kidney Cancer	4	85
6	Lung Cancer	5	102
7	Ovary	6	78
8	Non Cancer	7	37
Total data (sampel)			1065

Preprocessing Data



Gambar 3. Rancangan sistem normalisasi data

Sebelum digunakan sebagai *input feature* DNN, data *gene microarray* diproses terlebih dahulu. Jumlah *input feature* setiap sampelnya sebanyak 54675 baris. Dalam satu berkas untuk setiap sampel terdapat 2 kolom, dimana untuk kolom pertama sebagai variabel "ID_REF" dan kolom kedua "VALUE" sebagai nilai setiap variabel pada kolom pertama. Nilai setiap variabel yang terdapat pada setiap sampel memiliki *range* nilai yang bervariasi. Selain itu pada metode ini digunakan fungsi sigmoid dengan output {0…1} sebagai fungsi aktivasi dari setiap network yang terdapat dalam arsitektur DNN.

Normalisasi

Sebelum mengirimkan data ke jaringan DNN maka data dinormalisasi dengan *range* 0 sampai 1 menggunakan maksimal dan minimal data pelatihan [10]. Setelah proses normalisasi maka dataset dapat dikirim ke jaringan DNN untuk di *training*. Normalisasi data menggunakan metode *Min-Max Normalization*. Sebagai penyetaran nilai di setiap sampelnya, maka data dinormalisasikan dengan *range* {0…1}. Secara matematis normalisasi *Min-Max* dapat dijelaskan sebagai berikut :

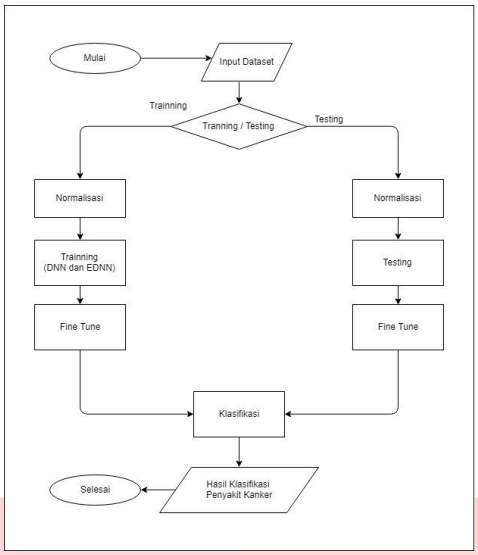
$$X' = \frac{(X - min)}{(max' - \min')}\ (max - min) + min'$$

Persamaan 18

dengan X' merupakan data baru, max'n min'bagai *range* maksimal dan minimal dari data yang dinormalisasi serta max dan min sebagai data terbesar dan terkecil sebelum normalisasi.

Diagram Alir Sistem Sekuensial

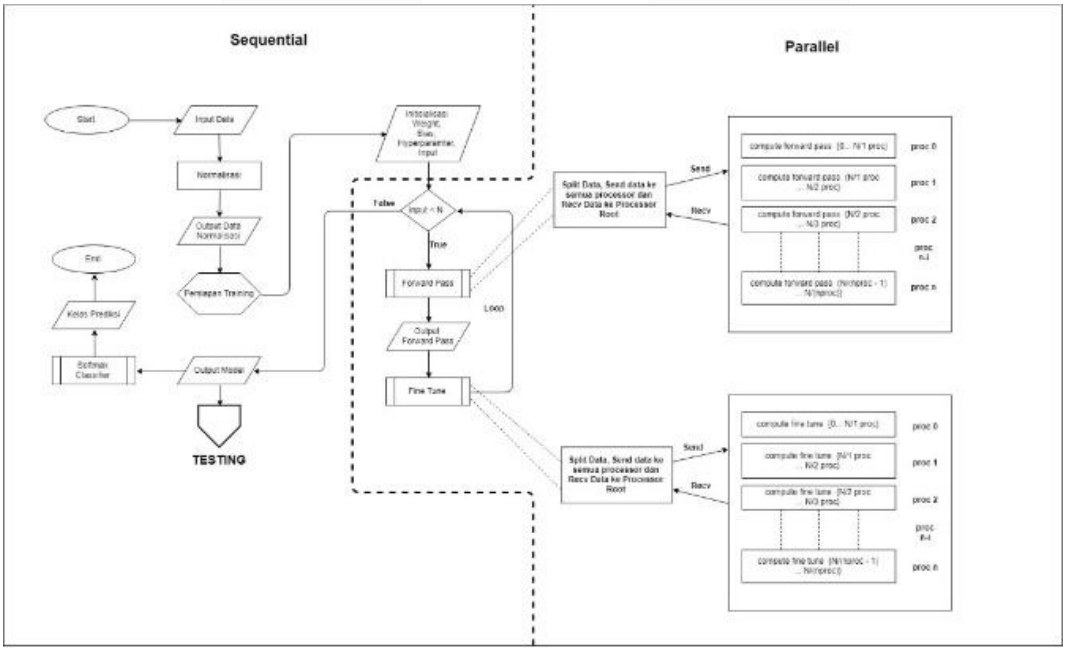
Berdasarkan Gambar 4 maka proses untuk memberikan inputan kepada sistem DNN harus menyesuaikan data inputan DNN. Setelah tahap *processing data* selesai, maka bentuk dari dataset yang menjadi *input features* DNN dilatih. Proses *learning* bergabung dengan SSAE untuk proses penguraian dimensi. Setelah diperoleh dimensi yang sesuai untuk proses *learning* maka output dari hasil SSAE menjadi inputan untuk pembelajaran pada jaringan DNN. Setelah proses *learning* selesai maka tahapan selanjutnya adalah klasifikasi data. Pada proses *testing*, dilakukan langkah yang berbeda dari proses *training*. Jika pada tahap *training* digunakan metode autoencoder, maka pada tahap testing hanya digunakan metode *multilayer perceptron* tanpa *fine tune backpropagation*.



Gambar 4. *Flowchart* sistem klasifikasi penyakit kanker dengan DNN.

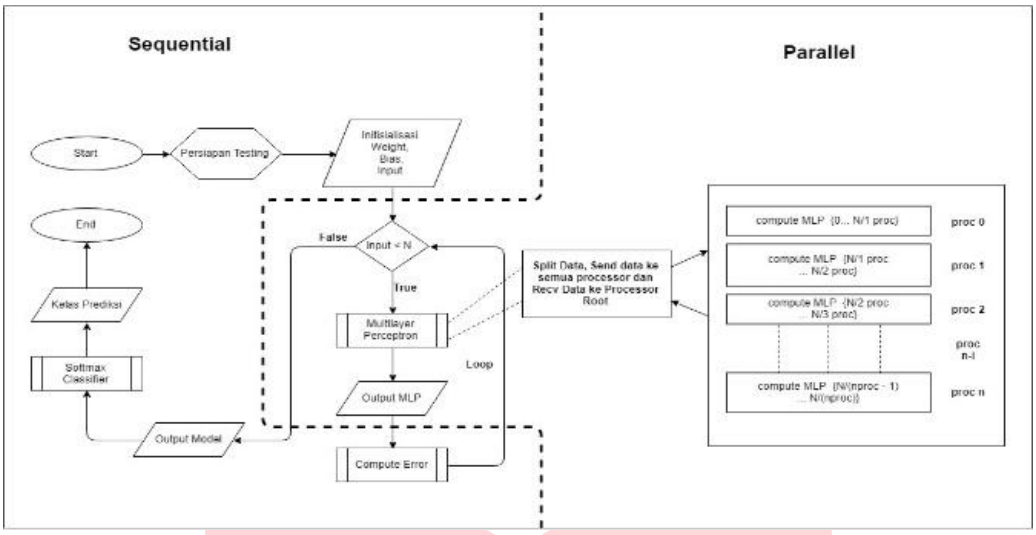
Diagram Alir Sistem Paralel

Dalam merancang algoritma paralel, harus dikonsep sedemikian rupa sehingga menghasilkan alur distribusi data paralel yang jelas. Dalam penelitian ini, terdapat beberapa konsep pengiriman data yang digunakan yaitu konsep persebaran data berdasarkan prosesor dan pengiriman data menyeluruh untuk setiap prosesor. Proses komputasi paralel yang dilakukan, dikerjakan secara bersama oleh setiap prosesor, dan dikembalikan ke proses utama. Pembagian komputasi paralel pada algoritma DNN ini terbagi atas dua bagian, yaitu pada tahap *training* dan *testing*. Pada tahap *training* paralelisasi digunakan untuk mengurangi waktu komputasi yang tinggi yaitu pada tahap *forward pass* dan tahap *fine tune*. Seperti flowchart pada gambar 3.5 yang mengambarkan alur paralel dan proses pengiriman data.



Gambar 5. *Flowchart* komputasi paralel tahap *training*

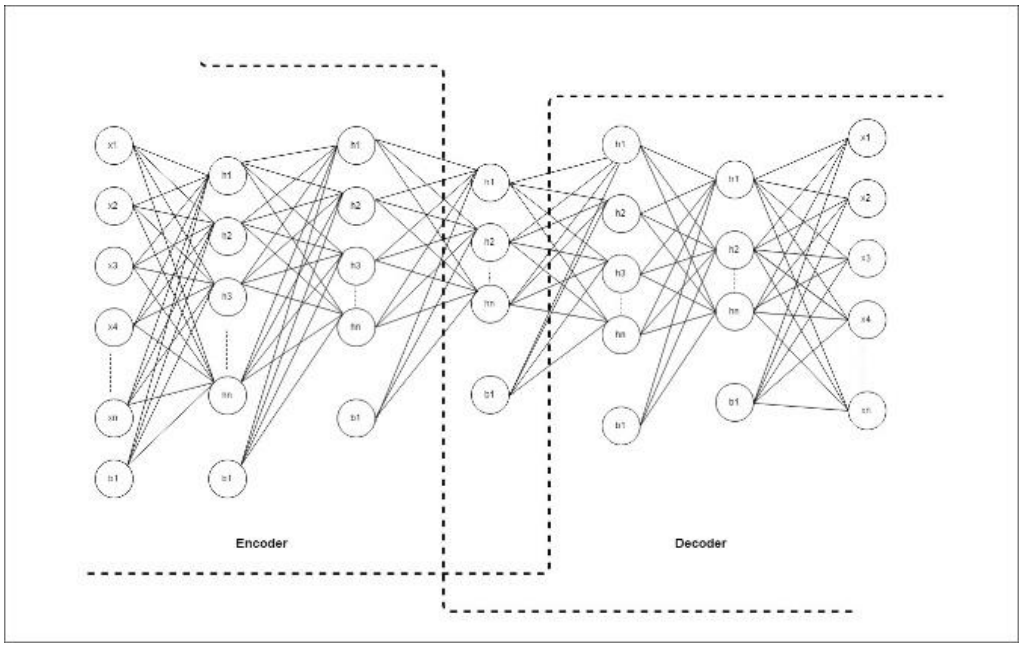
Beban komputasi besar pada algoritma DNN terdapat pada tahap training, tetapi pada tahap testing masih ada sebagain fungsi dengan beban komputasi cukup besar, sehingga dibutuhkan komputasi paralel untuk meringankan beban komputasi di tahap testing sehingga dapat meminumkan waktu eksekusi.



Gambar 6. Flowchart komputasi paralel untuk tahap testing algoritma DNN

Arsitektur DNN Autoencoder

Perancangan arsitektur untuk DNN merupakan tahap yang harus dilakukan. Dengan demikian, sistem dari DNN dapat di bentuk sesuai dengan kebutuhan pembelajaran yang akan di latih. Menentukan jumlah hidden layer dan neuron dari DNN merupakan bagian dari perancangan arsitektur. Semakin sesuai bentuk arsitektur yang dibuat dengan data yang akan dilatih, maka dapat mempengaruhi proses pembelajaran.



Gambar 7. Arsitektur Nerual Network

Seperti gambar 7 maka digunakan 54675 neuron untuk *input layer*, 500, 400, 300 neuron untuk encoder layer, 300, 400, 500 neuron untuk decoder layer dan 54675 neuron untuk reconstruction input sebagai rancangan untuk arsitektur DNN.

IV. Evaluasi

Penerapan dari sistem yang telah dirancang untuk melakukan proses pengujian merupakan tahapan wajib yang harus dilakukan pada sebuah penelitian. Implementasi dari sistem yang telah dirancangan untuk penelitian ini diterapkan menggunakan bahasa pemrograman *python* serta MPI sebagai API pemrograman paralel. Selain itu proses pengujian juga menggunakan *supercomputer* untuk menjalankan program DNN.

Komputasi paralel untuk algoritma DNN menggunakan paralelisasi CPU antar komputer yang saling bekerja dalam satu perintah dengan banyak data pemrosesan. Jenis *supercomputer* yang tepat untuk menjalankan algoritma ini adalah cluster CPU. Karena *hardware* dari *supercomputer* sangat mahal dan jarang digunakan untuk

keperluan pribadi, maka pengujian penelitian ini menggunakan fasilitas *supercomputer* yang terdapat di *High Performance Computing Laboratory* (HPC Laboratory), Fakultas Informatika, Telkom University.

HPC Laboratory memiliki 2 tipe *supercomputer* yang berbeda yaitu (a). Cluster Kucing berbasis GPU dan CPU, dan (b) Cluster Brokoli berbasis *high processor* dan *high memory*. Setiap jenis cluster memiliki keunggulan yang berbeda, tetapi untuk khasus penelitian ini *supercomputer* yang digunakan yaitu Cluster Brokoli dimana memiliki spesifikasi jumlah *core* sebanyak 48 *core* dan 2 *thread* untuk setiap *core* dengan total 96 *number of processor* (**np**). Adapun spesifikasi *supercomputer* yang digunakan akan dijelaskan pada tabel 4.1 sebagai bahan perbandingan untuk performansi komputasi.

Hasil Pengujian Skenario I

Berdasarkan skenario pengujian pertama untuk menentukan pengaruh dari jumlah data terhadap hasil dari akurasi klasifikasi, maka hasil dari pengujian dipresentasikan beberapa pengujian pada tabel 4.7 dan untuk pengujian penuhnya terlampir.

Tabel 2. Hasil skenario percobaan pertama

Kelas	Akurasi Sub Skenario (%)							
	1		3		6		10	
	train	**test**	**train**	**test**	**train**	**test**	**train**	**test**
0	96.6	80.0	98.7	94.1	99.3	97.3	99.6	98.1
1	94.2	80.0	98.2	94.1	99.1	97.3	99.2	97.7
2	94.2	80.0	97.8	94.1	98.7	97.3	99.2	97.7
3	94.2	80.0	98.2	94.1	98.7	97.3	99.2	97.7
4	94.2	80.0	98.2	94.1	98.7	97.3	99.2	98.1
5	94.2	80.0	98.2	94.1	98.7	97.3	99.2	98.4
6	92.9	80.0	97.8	94.1	98.7	97.3	99.2	98.4
7	97.0	80.0	98.7	94.1	99.3	97.3	99.6	98.4
Acc	**81.2**	**50.0**	**93.3**	**80.0**	**95.8**	**90.0**	**97.3**	**92.6**

Hasil Pengujian Skenario II

Pengujian kedua yaitu untuk mengetahui pengaruh dari *hyperparameter* terhadap akurasi dari *training* algoritma DNN. Pada pengujian kedua ini dilakukan 15 kali pengujian setiap *hyperparameter* disetiap percobaannya. Dengan ini didapatkan hasil terbaik dari percobaan yang uraikan pada tabel berikut ini.

Tabel 3. Hasil skenario percobaan kedua

No	Training Acc (%)	Testing Acc (%)	α	λ	ρ	β
1	95,25	88,24	0.05	0.00005	0.01	0.03
2	95.25	88,24	0.05	0.00005	0.02	0.03
3	95.25	88,24	0.05	0.00005	0.01	0.03
4	82.25	52.21	0.01	0.00001	0.05	0.03

Hasil Pengujian Skenario III

Pengujian ketiga yaitu membandingkan beban komputasi dan waktu komputasi anatara komputasi sekuensial dan paralel. Pada pengujian 3 jumlah data yang digunakan sebanyak 400 sampel *training* dan 136 sampel testing. Dalam pengujian ketiga dilakukan sebanyak 16 kali percobaan dengan variasi number of processor yang berbeda beda, maka hasil dari pengujian ketiga dideskripsikan pada tabel dibawah ini.

Tabel 4. Hasil skenario pengujian ketiga

Numbe of Processor	*Training* (menit)		*Testing* (menit)	
	Semua data	Epoch	Semua data	Iterasi
Serial	1631,14	4.077	53.22	0.391
2	946.01	2.365	23.24	0.170
4	492.72	1.231	14.81	0.108
6	342.01	0.855	10.28	0.075
8	280.59	0.701	7.90	0.058
10	227.38	0.568	6.67	0.049

12	204.10	0.510	5.91	0.043
14	174.00	0.435	5.20	0.038
16	166.84	0.413	4.92	0.036
18	148.35	0.371	4.53	0.033
20	137.93	0.344	4.31	0.031
22	131.97	0.329	4.03	0.029
24	125.14	0.312	3.80	0.027
26	133.78	0.347	4.55	0.033
28	140.77	0.351	4.89	0.036
30	264.67	0.651	9.12	0.067

Analisis Pengujian Skenario I

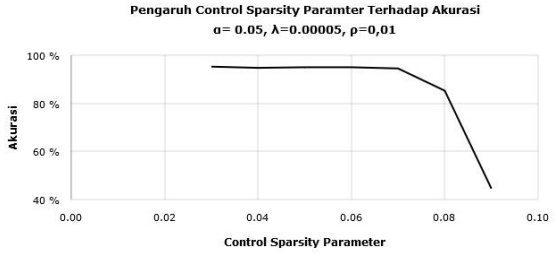
Telah menjadi sebuah kepastian bahwa semakin banyak waktu pembelajaran maka sistem DNN akan semakin akurat dalam menguji data selanjutnya. Ini berbanding lurus dengan jumlah data. Namun pada pengujian ini bertujuan melihatkan tren perkembangan akurasi pembelajaran. Akan ada titik jenuh dari proses pembelajaran. Tidak selamanya proses pembelajaran menggunakan banyak data *training* membuat sistem semakin akurat atau bisa jadi stagnan dan mungkin bisa saja menurun. Sebagai gambarannya disajikan data uji pertama dalam bentuk grafik untuk melihat tren perkembangan proses pembelajaran dari DNN.

Gambar 8. Grafik pengaruh jumlah data terhadap akurasi dari *training*

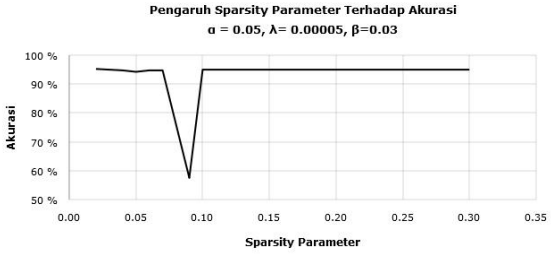
Berdasarkan hasil pengujian pada gambar 8 untuk akurasi *training*, maka dapat jelaskan bahwa semakin banyak data *training* peningkatan akurasi semakin besar Pada saat menggunakan 80 sampel data *training* akurasi hanya berkisar di 81,2 % dan mengalami peningkatan yang cukup tinggi pada saat data *training* yang digunakan 4 kali lebih banyak dari pengujian pertama. Akurasi saat data yang digunakan sebanyak 320 yaitu 94,4 %, namun setelah data di tambahkan lebih banyak lagi maka akurasi hanya bertahan pada *range* 95% sampai 97 %. Puncak akurasi tertinggi dalam pengujian ini yaitu 97,3 % dengan jumlah data *training* sebanyak 788 sampel dengan 8 kelas klasifikasi.

Analisis Pengujian Skenario II

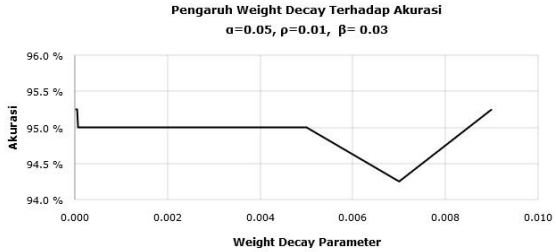
Hyperparameter DNN sangat mempengaruhi proses pembelajaran. Untuk inisialisasi DNN, hubungan antara 4 *hyperparameter* yaitu *learning rate*, *weight decay parameter*, *sparsity*, *dan control sparsity* sangat berhubungan satu sama lainnya. Sementara itu tujuan pengujian kedua ini yaitu melihat seberapa besar pengaruh setiap *hyperparameter* ini terhadap proses pembelajaran DNN.



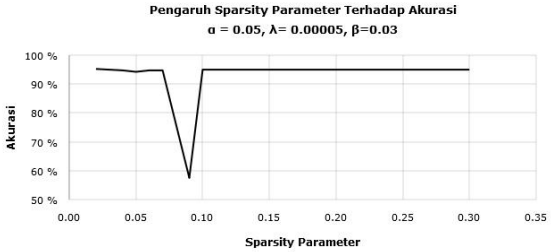
Gambar 9. Grafik pengaruh *Control Sparsity Parameter* Terhadap Akurasi



Gambar 11. Grafik pengaruh *Sparsity Parameter* Terhadap Akurasi



Gambar 10. Grafik pengaruh *Weight Decay Parameter* Terhadap Akurasi



Gambar 12. Grafik pengaruh *learning rate* terhadap akurasi

Berdasarkan gambar 10, pengaruh weight decay parameter terhadap akurasi tidak terlalu siginifikan berpengaruh. Dapat dilihat bahwa semakin tinggi weight decay parameter tidak selalu menaikan tingkatan akurusi. Penurunan akurasi terjadi saat nilai dari parameter ini pada *range* 0.006 sampai 0,008 memperoleh akurasi dibawah 94.5 %. Walaupun dengan menggunakan parameter tersebut masih mendapatkan akurasi tinggi, tetapi dibandingan dengan nilai lainnya ini termasuk sebuah penurunan terhadap proses *learning*. Berdasarkan hasil pengujian weight decay parameter terbaik yaitu 0.00005 dengan akurasi *training* mencapai 95.25 %.

Selain *weight decay parameter*, *control sparsity* juga termasuk salah satu *hyperparameter* dari DNN. Berdasarkan gambar 9 dapat dilihat bahwa ini mempengaruhi proses pembelajaran. Ini dapat dilihat ketika nilai dari parameter ini diset mendekati 1 dimana akurasi semakin menurun. ketika nilai diset 0.09 menurun ke 50,75%. Semetara itu nilai dari parameter yang lebih tinggi dari 0.8 memperoleh akurasi lebih rendah dari sebelumnya. ini membutikan bahwa parameter ini berpengaruh terhadap hasil dari akurasi *training*. Dimana dengan percobaan ini diperoleh parameter terbaik untuk *control sparsity parameter* yaitu pada nilai 0.03

Dari dua parameter yang telah dibahas, kedua parameter tersebut terlihat memiliki perbengaruh terhadap akurasi. Sementara itu sparsity parameter juga terlihat berpengaruh terhadap akurasi, karena pada teorinya, sparsity digunakan untuk mengatur informasi dari setiap input yang masuk ke jaringan. Berdasarkan dari grafik yang pada gambar 11, maka dapat diamati bahwa nilai dari sparsity memberikan dampak yang berpengaruh terhadap hasil dari akurasi jika parameter diset dengan nilai 0.09 dan memperoleh akurasi *training* dibawah 60%. Tetapi jika parameter diset selain nilai itu maka hasil dari akurasi memberikan nilai yang terbaik, dimana hasil akurasi tertinggi diperoleh sekitar 95,25 % dengan nilai parameter 0.01.

Pada penelitian ini nilai dari learning rate yang tinggi akan memberikan akurasi terbaik. Seperti dapa grafik berikut bahwa nilai dari akurasi ketika learning rate diset 0.05 lebih baik dari pada 0.03 dengan akurasi sekitar 20 % dan lebih baik lagi dari nilai learning rate yang diset 0.01 yang hanya memperoleh akurasi 82 %. Sekilas pengaruh dari learning rate memberikan dampak terhadap pembelajaran. Namun tidak selamanya learning rate yang tinggi akan memberikan akurasi yang baik. Karna berdasrkan teorinya learning rete merupakan langkah pembelajarnnya. Semakin cepat langkah pembelajaran tidak selamanya proses training akan semakin baik. Karena bisa saja dengan langkah yang besar akan melawati tahap minimum lokal terbaiknya.

Hubungan yang erat antara semua *hyperparameter* memberi pengaruh terhadap hasil dari akurasi, berdasarkan tabel 5 tersebut maka hasil dari percobaan terbiak yaitu pada pengujian 1 dan 2 dengan masing masing memperoleh akurasi 95 % lebih tinggi dari yang lainnya. Sementara itu pada pengujian 6 dan 7 hasil dari akurasi yang diperoleh sangat rendah. Jika pengaruhnya dilihat dari segi hypermeter, maka untuk mendapatkan akurasi yang terbaik dapat menggunakan *hyperparameter* dengan $\boldsymbol{\alpha}$ 0.05, $\boldsymbol{\lambda}$ 0.00005, $\boldsymbol{\rho}$, 0.01 dan $\boldsymbol{\beta}$ 0.5 atau menggunakan $\boldsymbol{\rho}$ yang berbeda yaitu 0.035. Dari informasi ini, maka pengaruh dari *hyperparameter* terjadi ketika semua nilai dari parameter disetel dengan nilai terbaik. Artinya setiap *hyperparameter* ini saling berketergantungan dan tidak dapat dinilai satu persatu.

Analisis Pengujian Skenario III

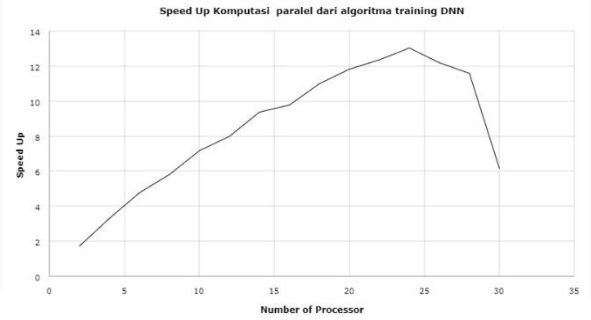
Sebagai tujuan utama dalam penelitian ini, analisis hasil paralelisasi dibutuhkan untuk memberikan solusi dari lamanya waktu komputasi algoritma dari DNN. Berdasarkan skenario pengujian ketiga, maka hasil analisis speed up, efisiensi dan speed up berdasarkan *Amdahl law* di uraikan berdasarkan tabel berikut ini :

Tabel 6. Hasil dari analisis performansi paralel.

Number of Processor	Waktu eksekusi	*Speed up*	Efisiensi (%)	Amdahl law speed up
Serial	1631,14	-	-	-

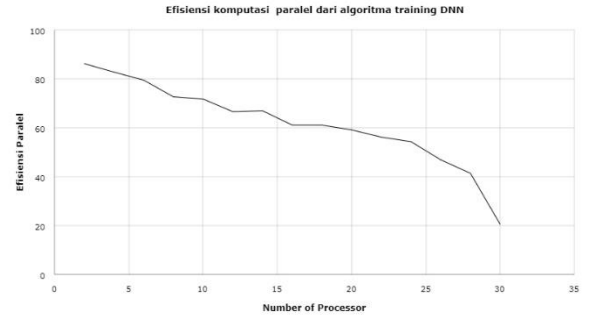
2	946.01	1.72	86.21	1.63
4	492.72	3.31	82.76	2.40
6	342.01	4.77	79.48	2.85
8	280.59	5.81	72.66	3.14
10	227.38	7.17	71.73	3.35
12	204.10	7.99	66.59	3.50
14	174.00	9.37	66.95	3.62
16	166.84	9.77	61.10	3.72
18	148.35	10.99	61.08	3.79
20	137.93	11.82	59.12	3.86
22	131.97	12.35	56.17	3.91
24	125.14	13.03	54.30	3.96
26	133.78	12.19	46.89	4.00
28	140.77	11.58	41.38	4.03
30	264.67	6.16	20.54	4.06

Dari segi kecepatan waktu komputasi, hasil dari penelitian ini membuktikan bahwa komputasi paralel terhadap *training* dari algoritma DNN berhasil untuk mempercepat waktu komputasi. *Speed up* komputasi paralel tertinggi terjadi saat algoritma di kerjakan menggunakan 24 *np* dengan speed up 13,03 kali lebih cepat dari algoritma sekuensial. Jika dilihat dari waktu komputasi, saat algoritma dikerjakan menggunakan 24 np waktu komputasi yang diperoleh sekitar 125 menit dan waktu komputasi sekuensialnya sekitar 1631 menit. Tetapi pada pemrosesan kali ini algoritma dari DNN mengalami titik jenuh di pengerjaan 26 np, dimana pada percobaan ini *speed up* menurun dibandingkan saat menggunakan 24 np. Pada gambar 13 ini terlihat semakin menurun untuk jika di kerjakan lebih dari 24 np.



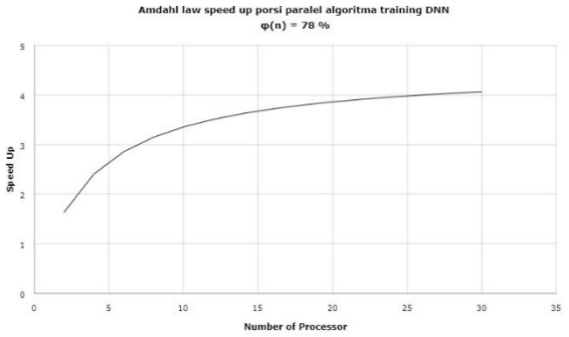
Gambar 13. Grafik Speed up komputasi paralel algoritma DNN.

Analisis performansi komputasi paralel tidak hanya sebatas speed up antara waktu sekuensial dan waktu paralel. Selain dari itu, efisiensi juga berpengaruh terhadap analisis dari sebuah algoritma paralel. Berdasarkan hasil pengujian, efisiensi tertinggi terjadi saat algoritma *training* DNN dikerjakan dengan 2 np. Ini menandakan bahwa biaya komputasi yang digunakan saat pemrosesan lebih tinggi jadi efisiensi dari pemrosesan paralel terlihat lebih tinggi. Secara keseluruhannya efisiensi dari paralel terlihat menurun. Ini dikeranakan semakin banyak np yang digunakan maka semakin sedikit biaya komputasi yang diperlukan. Penurunan ini dapat berdasarkan grafik dibawah ini.



Gambar 14. Grafik *speed up* komputasi paralel algoritma DNN.

Algoritma ini dirancang dengan porsi paralel sebanyak 78%, dan 22 % porsi dari komputasi sekuensial. Dengan jumlah porsi paralel yang diketahui ini, menggunakan persamaan *Amdahl law* maka *speed up* terkecil dari komputasi paralel dapat di estimasi. Berdasarkan hasil dari pengujian dengan menggunakan 2 np, diperoleh *speed up* sekitar 1.72 dengan estimasinya berdasarkan *Amdahl law* lebih dari 1.63. Sedangkan untuk estimasi untuk penggunaan 24 np sekitar 4.03. Dengan ini, *speed up* dapat di estimasi sebelum melakukan proses komputasi. Untuk gambaran lengkap peningkatan *speed up* berdasarkan *Amdahl law* dapat disajikan berdasarkan gambar dibawah ini.



Gambar 15 Estimasi speed up *Amdahl law*.

V. Kesimpulan

Pada penelitian ini, *deep learning* dapat memberikan solusi untuk klasifikasi data *gene expressrion* menggunakan DNN. Ini dapat dilihat dari berbagai skenario pengujian yang dilakukan. SSAE bekerja dengan baik dalam proses pembelajaran klasifikasi gen kanker. Menggunakan *hyperparameter* yang terbaik, akurasi dari pembelajaran dapat ditingkatkan.

Maka dengan ini, banyak yang dapat disimpulkan dari hasil penelitian ini yaitu tren menggunakan data *training* yang banyak akan meningkatkan akurasi dari *training* seperti dari pengujian pertama akurasi dari *training* diperoleh sekitar 97,3 % . Tetapi penggunaan data *training* yang terlalu banyak tidak akan membuat akurasi menjadi sempurna. Ini terlihat ketika data yang digunakan sekitar 400 sampai dengan 788 sampel hanya stagnan di posisi akurasi 90 sampai 98 %.

Selain itu *hyperparameter* juga berpengaruh terhadap proses pembelajaran, ini terlihat dari pengujian kedua. Tetapi hubungan antar *hyperparameter* sangat kuat. Ini tidak dapat diukur dengan satu persatu. Melainkan harus diukur secara bersamaan untuk menemukan *hyperparameter* terbaik. Dari pengujian kedua ini diperoleh *hyperparameter* terbaik dengan α 0.05, λ 0.00005, ρ, 0.01 dan β 0.5 atau menggunakan ρ yang berbeda yaitu 0.035.

Pada segi komputasi paralel, algoritma terlihat sangat bekerja dengan baik. Ini dibuktikan saat algoritma training dikerjakan menggunakan 24 np. *Speed up* yang diperoleh sekitar 13.03 lebih tinggi dari percobaan yang lainnya. Tetapi tampak lebih rendah jika algoritma dikerjakan lebih dari 24 np. Sebenarnya ini berhubungan dengan jumlah *core* dan *thread* pada fisik server digunakan. *Supercomputer* yang digunakan hanyak memiliki 24 *core* dalam 1 servernya. Tetapi 1 *core* pada server itu memiliki 2 *thread* sehingga dalam satu server dapat mengerjakan 48 proses namun akan terlihat lebih lambat karena proses telah dibagi menjadi 2. Solusi dari ini dapat menggunakan pemrograman *hybrid* antara MPI dengan OpenMP. Kerena OpenMP dapat melakukan proses komputasi dalam pemrosesan 1 prosesor dengan banyak *threard*.

DAFTAR PUSTAKA

[1] Pusat data dan informasi. 2015. Situasi penyakit kanker: Buletin Jendela Data Informasi Kesehatan. Jakarta: Kementrian Kesehatan Republik Indonesia. ISSN 2088 – 270X

[2] Y. Yuan, Y. Shi, C. Li, J. Kim, W. Cai, Z. H, F. David Dagan. 2016. DeepGene: an advanced cancer type classifier based on deep learning and point mutations. The 27th International Conference on Genome Informatics.

[3] Schmidhuber, J. 2015. Deep Learning in Neural Network: An Overview. Neural Network. 61: 85 -117.

[4] Olshausen, B.A. 1996. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature. Vol 381. Issue 6583 : 607-609

[5] kurt. H, Maxwell. S, Halbert. W. 1989. Multilayer feedforward network are universal approximators. Neural Networks. Vol 2. Issue 5: 359-366. ISSN 0893-6080.

[6] Min. S, Lee. B, Yoon. S. 2017 .Deep Learning in Bioinformatics. Briefings in Bioinformatics. Vol 18. Issue 5: 851-869.

[7] Fakoor. R, Ladhak. F, Nazi. A, Huber, M. 2013. Using Deep Learning to Enhance Cancer Diagnosis and Classificaiton. International Confference on Machine Learning, Vol 28.

[8] Kingma. DP, Welling. M. 2013. Auto-Encording Variational Bayes. ArXiv e-prints. Vol 10.
[9] Ilham, M. 2017. Klasifikasi Sinyal ECG menggunakan Deep Learning dengan Stacked Denoising Autoencorders. Universitas Telkom.
[10] S, Vikas. B, Nikhil. Rahul K, Sevakula. Nishchal K, Verma. C. Yan. 2016. Layerwise feature selection in stack spare autoencoder for tumer type prediction. IEEE
[11] Cortes, C. Vapnik, V. 1995. Support Vector Networks. Machine Learning. 20 (3) : 273 – 297.
[12] Biau, G. 2012. Analysis of a random forests model. Journal of Machine Learning Research vol 13, pp 1063-1095.
[13] Goodfellow, I. Bengio, Y. Courville, A. 2016. Deep Learning. MIT Press
[14] Mitchell, Melanie. 1996. An Introduction to Genetic Algorithms. MIT Press.
[15] Miikkulainen, R, dkk. 2017. Evolving Deep Neural Network. Sentient Technologies and The University of Texas at Austin. arXiv:1703548v2
[16] Dufourq, E. A Bassett, B. 2017. EDEN: Evalutionary Deep Network for Efficient Machine Learning. arXiv:1709.09161v1
[17] G, Celine. 2014. Deep Learning Via Stacked Sparse Autoencoder for automated Voxel-wise Brain parcellation Based on Functional Connectivity. The School of Graduate and Postdoctoral Stuides, The University of Western Ontario.